ARMY RESEARCH LABORATORY


# A Scalable and Dynamic Testbed for Conducting Penetration-Test Training in a Laboratory Environment

**by Jaime C Acosta, Scott Freeman, and Felipe Sotelo**

**ARL-TR-7236** **March 2015**

## NOTICES

### Disclaimers

The findings in this report are not to be construed as an official Department of the Army position unless so designated by other authorized documents.

Citation of manufacturer’s or trade names does not constitute an official endorsement or approval of the use thereof.

Destroy this report when it is no longer needed. Do not return it to the originator.

**Army Research Laboratory**
White Sands Missile Range, NM 88002-5513

**ARL-TR-7236** **March 2015**

# A Scalable and Dynamic Testbed for Conducting Penetration-Test Training in a Laboratory Environment

**Jaime C Acosta, Scott Freeman, and Felipe Sotelo**
**Survivability/Lethality Analysis Directorate, ARL**

# REPORT DOCUMENTATION PAGE

*Form Approved*
*OMB No. 0704-0188*



| 1. REPORT DATE *(DD-MM-YYYY)* | 2. REPORT TYPE | 3. DATES COVERED (From - To) |
|---|---|---|
| March 2015 | Final | June 2014–October 2014 |

**4. TITLE AND SUBTITLE**
A Scalable and Dynamic Testbed for Conducting Penetration-Test Training in a Laboratory Environment

**5a. CONTRACT NUMBER**

**5b. GRANT NUMBER**

**5c. PROGRAM ELEMENT NUMBER**

**6. AUTHOR(S)**
Jaime C Acosta, Scott Freeman, and Felipe Sotelo

**5d. PROJECT NUMBER**

**5e. TASK NUMBER**

**5f. WORK UNIT NUMBER**

**7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES)**
US Army Research Laboratory
ATTN: RDRL-SLE-A
White Sands Missile Range, NM 88002-5513

**8. PERFORMING ORGANIZATION REPORT NUMBER**
ARL-TR-7236

**9. SPONSORING/MONITORING AGENCY NAME(S) AND ADDRESS(ES)**

**10. SPONSOR/MONITOR'S ACRONYM(S)**

**11. SPONSOR/MONITOR'S REPORT NUMBER(S)**

**12. DISTRIBUTION/AVAILABILITY STATEMENT**
Approved for public release; distribution unlimited.

**13. SUPPLEMENTARY NOTES**

**14. ABSTRACT**
This report describes a novel testbed for training penetration testers who evaluate the security of computer networks. This testbed was generated based on an informal study of the common needs of real-life penetration testers. Only open-source technologies are used and step-by-step instructions are provided on how to create the testbed. A case study is also included with a sample scenario and the steps that a penetration tester may take to evaluate a network. Lastly, the report includes lessons learned and limitations of the testbed.

**15. SUBJECT TERMS**
IA testbed, penetration testing, security training, pentest

| 16. SECURITY CLASSIFICATION OF: | | | 17. LIMITATION OF ABSTRACT | 18. NUMBER OF PAGES | 19a. NAME OF RESPONSIBLE PERSON |
|---|---|---|---|---|---|
| a. REPORT | b. ABSTRACT | c. THIS PAGE | | | Jaime C Acosta |
| Unclassified | Unclassified | Unclassified | SAR | 40 | 19b. TELEPHONE NUMBER (*Include area code*) (575) 678-8115 |

## Contents

**List of Figures** v

**List of Tables** vi

**1. Introduction** 1

**2. Methodology** 3

2.1 Eliciting Needs ....3

2.2 Platform Selection ....4

2.3 Building the Testbed ....5

2.3.1 XenServer Installation ....5

2.3.2 CORE Installation ....5

2.3.3 XenServer and CORE Integration ....6

2.3.4 Virtual Ethernet Node Setup ....10

2.3.5 XenServer Startup and Shutdown from within CORE ....11

2.3.6.1 XenServer Login Note ....13

2.3.6.2 XenAPI.py ....13

**3. Case Study—Building an Exploitable Scenario** 13

3.1 The CORE scenario ....14

3.2 Setting up FTP on a XenServer VM ....14

3.3 Assigning the FTP Server to a Subnetwork in CORE using DHCP ....16

3.4 Executing an FTP Client ....17

3.5 Penetration-Test Walkthrough ....17

**4. Lessons Learned** 18

4.1 Adding Interfaces for Use with CORE RJ-45 Components ....18

4.1.1 Failed Attempts at Adding Multiple RJ-45 Components Bound in CORE ....19

4.1.2 Implemented Solution ....20

4.2 Broadcast Packets Are Not Isolated ....21

4.2.1 Implemented Solution ....21

**5. Conclusions** 21

**6. References** **23**

**Appendix: Contents of Startup, Shutdown, and FTP-Client Scripts** **27**

Startup Script ..... 28

Shutdown Script ..... 29

FTP client script: conn.sh ..... 30

**List of Symbols, Abbreviations, and Acronyms** **31**

**Distribution List** **32**

## List of Figures


Fig. 1 CORE-downloads Website....5
Fig. 2 The CORE RJ-45 Component....6
Fig. 3 Network devices on XenServer virtual machine....7
Fig. 4 Adding a network device to a virtual machine in XenServer....7
Fig. 5 Router node options selection menu....8
Fig. 6 Router node-interface configuration screen....8
Fig. 7 Router node-services configuration screen....9
Fig. 8 CORE router-node DHCP configuration....9
Fig. 9 CORE RJ-45 component options menu....10
Fig. 10 RJ-45 component interface-selection menu....10
Fig. 11 Kali Linux network-manager interface list....11
Fig. 12 CORE-session menu....11
Fig. 13 CORE-session hooks menu....12
Fig. 14 CORE hook-script configuration script....12
Fig. 15 CORE hook-script shutdown script....12
Fig. 16 CORE-configuration script content....13
Fig. 17 CORE-shutdown script content....13
Fig. 18 CORE-network scenario....14
Fig. 19 Creating 2 accounts in the EasyFTP server GUI....15
Fig. 20 Starting the Easy FTP daemon....15
Fig. 21 Startup shortcut to FreeFTPd....16
Fig. 22 Runtime snapshot for a Windows FreeFTPd server in XenServer....16
Fig. 23 DHCP configuration file on a CORE router node....17
Fig. 24 Two CORE RJ-45 components bound to the same physical interface....19
Fig. 25 A screenshot of some of the interfaces created by CORE....19
Fig. 26 A screenshot of the interface created for node n3 in CORE....20
Fig. 27 The bridges created by CORE....20

## List of Tables

Table Technologies chosen to satisfy needs.....4

# 1. Introduction

A critical step in evaluating the security posture of a computer network is the penetration test. The quality of the results of a penetration test is dependent on the knowledge and expertise of the human evaluator. To ensure that evaluators have adequate skills to perform penetration tests, there are regulations in place that must be satisfied before personnel are admitted to the evaluations. For example, Department of Defense (DOD) Directive 8570[1] describes requirements that technical personnel must meet to be allowed to conduct information-assurance testing of US military systems. The regulations typically require that specific third-party certifications are obtained and maintained through yearly fees and continuing professional education by the testers involved. Among these third parties and their certifications are the International Information Systems Security Certification Consortium, Inc., ISC2's Certified Information Systems Security Professional[2] (CISSP); Information Systems Audit and Control Association (ISACA) Certified Information Systems Auditor[3] (CISA); and International Council of Electronic Commerce Consultants (EC-Council) Certified Ethical Hacker[4] (CEH).

While these certifications provide a sort of "rite of passage" it is still important that security evaluators learn about the latest vulnerabilities and proactively practice to improve the skill sets needed for successful penetration tests (pentests). Here are some steps toward developing and improving a pentest skill set:

1. Identify a vulnerability
2. Set up a practice testbed in which to execute the vulnerable software
3. Develop or procure an exploit against the vulnerability
4. Test the exploit in the vulnerable software's environment

To identify new or zero-day vulnerabilities, a tester can obtain the software of interest and then execute a combination of actions such as reverse engineering, debugging, and fuzz testing. A vulnerability usually is found by giving the software an unexpected input, causing a system crash, and then tracing the reason for the crash. Alternatively, testers can look up known vulnerabilities on public databases such as the National Institute of Standards and Technology's National Vulnerability Database (NVD)[5] and MITRE's Common Vulnerabilities and Exposures.[6] These databases inform the information-security community of vulnerabilities that should be tested when at-risk software is encountered. These databases serve the user community by alerting it of potential risks associated with vulnerable software. Lastly, the databases identify security issues that must be patched in order to decrease the potential for breaches.

To set up practice testbeds, testers must have at their disposal machines and software that are capable of running the vulnerabilities and exploits. Virtualization software such as VirtualBox,[7] VMware products,[8] and XenServer products[9] facilitate the management of these collections of machines and exploits. This virtual environment usually consists of several versions of operating systems (e.g., several instances of Windows 7, each with different service packs and security patches). This environment is isolated from the any external networks—unless it is part of a "honeypot system" [10] aimed at collecting information about known threats. This isolation eliminates the risk of an external attacker exploiting the vulnerability in the practice environment.

After the vulnerability is identified and a suitable environment has been created, the next step for a tester is to develop or otherwise procure an exploit. Exploits for zero-day vulnerabilities are developed using the same methods for identifying vulnerabilities: reverse engineering, debugging, and fuzz testing. Publicly known exploits are also available from online databases such as Offensive Security's Exploit DB[11] and Rapid 7's Vulnerability and Exploit database.[12]

The purpose of the exploit is to use the vulnerability as an entry point through which to execute a payload to accomplish a higher-level goal: executing arbitrary code, escalating privileges, pivoting, eavesdropping, denial of service, etc. The Metasploit[13] toolkit automatically generates payloads that can be embedded into exploits to create any of these higher-level goals. In some cases, these payloads are publicly known and have been added to signature detectors such as antivirus and intrusion-detection systems. To circumvent these technologies, tools such as Veil[14], unicorn.py[15], and UPX[16] are used to modify the exploit and payload signatures by applying compression and encryption to the binary data.

Lastly, to test the exploit against the vulnerability, a tester usually executes the attack through the same machine that contains the vulnerability (in the case of a local exploit) or from a second remote machine (in the case of a remote exploit). While this learning exercise provides a tester with practice on developing and executing exploits, it does not resemble real field scenarios. Network systems undergoing penetration testing usually consist of several hosts, networking components (routers, switches), and people. There is a gap between real field tests and the small practice scenario described above. The penetration tester must consider several real-world variables including the network system's susceptibility to social engineering and colluding attackers and whether multiple vulnerabilities exist.

One way the security community practices in field-like scenarios is through "capture the flag" events, which are usually held as part of training (such as that offered by InfoSec[17]) and security conferences (e.g., DEFCON[18]) and in academia[19,20] as well as government and industry[21,22]. However, these scenarios are difficult to set up; moreover, after the events there is no way for participants to re-attempt the scenarios. In addition, during these events the network is tainted by the actions of competing testers, which is not characteristic of most real-life field tests. Other testbeds such as DeterLab[23] and HackaServer[24] are available to the public but are only remotely

accessible. In the case where sensitive information is present, these would not be suitable for testing. This technical report describes a novel way to create capture-the-flag scenarios that resemble field events. Our contributions are the following:

1. A method to set up several scenarios using only publicly available, open-source software. These environments enable several analysts to practice on scenarios independently (each analyst can start their own isolated scenario instance) and simultaneously using the same underlying hardware.
2. Implementation details on how to use the common open research emulator (CORE)[25] to develop dynamic network topologies for both tactical and strategic military networks. This eliminates the need for physical network components; CORE combined with the Extendable Mobile Ad Hoc Network Emulator (EMANE)[26] can emulate the entire network stack (physical to application-layer protocols).

## 2. Methodology

To build a dynamic testbed, our team 1) elicited needs from real penetration testers, 2) selected a platform for development, 3) built the dynamic environment, and 4) performed a case study to evaluate advantages/disadvantages and other "lessons learned". Steps 1–3 are detailed in this section; the case study is examined in Section 3.

### 2.1 Eliciting Needs

We informally asked several penetration testers in the US Army Research Laboratory (ARL) for their preferences, suggestions, and comments for an ideal testbed. From that input came this list of the most-needed capabilities:

A. Dynamic topology support: an easy way to create scenarios with varying topologies, e.g., a mix of wired and wireless/ad hoc and infrastructure networks.

B. Easily configurable: an environment that can be configured from a single point of entry. Ideally this would be a graphical interface with drag-and-drop capabilities.

C. Support for tactical technologies: a way to incorporate tactical Army networks including wireless sensor networks and mobile ad-hoc networks along with the protocols (especially at the network layer) associated with these.

D. Support for many operating systems: The testbed should be able to host Windows, Linux, MacOS, Android, and other operating systems without much effort.

E. A simple and automatic “restore” function: Many times during testing, scenarios become corrupt because of the nature of the tests. There should be a way to facilitate system restore, not only for single systems but for all hosts involved in the scenario.

F. Ability to execute/analyze malware: The environment should be realistic enough to allow malware to work as intended.

G. Ability to facilitate creation of multiple and varied capture-the-flag scenarios: Developing and modifying existing scenarios should be relatively easy—ideally through a graphical user interface (GUI).

H. Ability to simultaneously execute multiple isolated scenarios: If multiple users are running through scenarios on the same system, even if the scenarios are the same, each user should have an isolated scenario.

## 2.2 Platform Selection

Based on the needs described by the ARL penetration testers, we decided to use the CORE for managing scenario topology, services (e.g., server-side software like Secure Shell [SSH] servers), nodes running generic Linux installations, and any other scenario configuration. We also chose to use XenServer virtualization software to host nodes with other operating systems (Windows, MacOS, etc.). The Table describes how CORE and XenServer are used to satisfy the needs.

Table Technologies chosen to satisfy needs

| Need | Technology | Description |
|---|---|---|
| A | CORE | CORE enables users to create topologies through the GUI or through their Python application programming interface (API). CORE supports wireless and wired and provides software emulation for layers 3+ in the network stack. This means any software (including routing protocols) that is able to execute on Linux systems can be part of the scenarios. CORE is compatible with the EMANE plugins for Layer 1 and 2 communication models (e.g., custom military waveforms, 802.11 wireless, etc.). |
| B | CORE | CORE allows users to create topologies and configurations and then saves these to either an .imn or xml file. Configurations include custom startup behaviors for nodes as well as custom CORE scenario behavior (e.g., starting a XenServer virtual machine during or before execution). |
| C | CORE | CORE has built-in functionality to allow both infrastructure and mobile ad-hoc environments. Regarding mobility, these scripts can be generated with ScenGen (a well-known mobility script generator) and then imported into CORE. Linux-based network layer protocols are supported. |
| D | XenServer/CORE | CORE has a hardware-in-the-loop feature that can be tied with XenServer virtual machines, which can be used to host a wide range of operating systems. |
| E | XenServer/CORE | XenServer allows fast clone and snapshots that can be reverted in case of system malfunction. This functionality can be accessed through the back-end Python XenAPI.py by CORE to load (on scenario start) and revert (on scenario termination). |

| Need | Technology | Description |
|---|---|---|
| F | XenServer/CORE | CORE can be used to generate realistic network topologies and XenServer can host real operating systems. Malware behavior can be analyzed across subnetworks, operating systems, etc. |
| G | XenServer/CORE | CORE's scenario files (.imn or .xml) can be saved, loaded, and modified at any time. As long as the resources are still available (e.g., XenServer virtual machines have not been removed), the scenario will execute as initially configured. |
| H | XenServer | XenServer can bind a network to a physical Network Interface Card (NIC). If different networks are bound to different network interfaces, traffic will be isolated. |

## 2.3 Building the Testbed

### 2.3.1 XenServer Installation

XenServer 6.0.2 was installed on a Dell PowerEdge M820 Blade Server with a 2x IntelXeonE5-4603 2.00GHz processor and 8GB RDIMM memory. XenServer ISO builds are available on the Web.[27] XenServer was installed from the ISO with default configurations.

XenServer was managed through a Windows Laptop with the XenCenter Management Console software.

### 2.3.2 CORE Installation

To install CORE, a Debian-type Linux virtual machine was created in XenServer. We assigned 7 virtual network interfaces with this virtual machine. Next, a Kali-Linux 1.0.6 distribution[28] and the CORE software were installed from the source code available on the Web[29] (see Fig. 1).

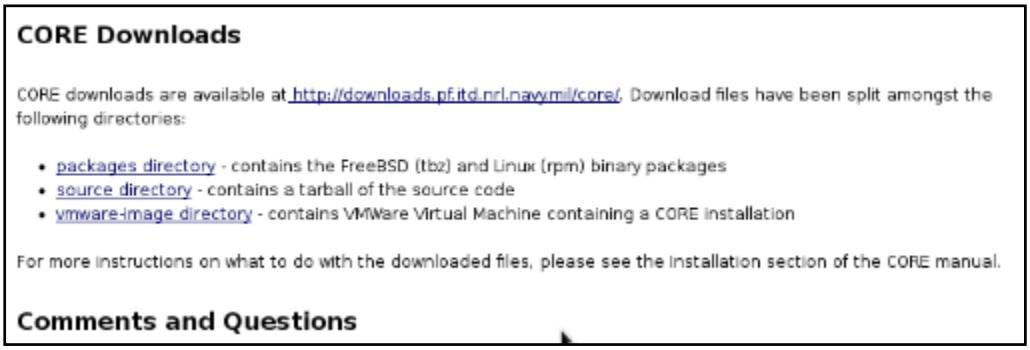


Fig. 1 CORE-downloads Website

The following commands were used to install CORE from sources along with all dependencies:

```
tar -zxvf CORE-4.6.tar.gz
cd CORE-4.6
apt-get update
apt-get install libev-dev
apt-get install bridge-utils
apt-get install ebtables
apt-get install libtk-img
./configure
make
make install
ln -s -T /etc/init.d/CORE-daemon S16CORE-daemon
service CORE-daemon start
```

The CORE includes a graphical interface to facilitate building networks. Network components (routers, switches, and generic Linux hosts) can be added from the toolbar on the left panel and then dropped onto the main canvas. Using the link tool, located below the start/stop button, virtual connections can be formed between any 2 nodes.

Virtual machines (VMs) running on XenServer can also be connected to the network. The RJ-45 component (highlighted in Fig. 2) can be used to communicate with external entities (e.g., external hardware and other virtual machines) through the host-network interface card.

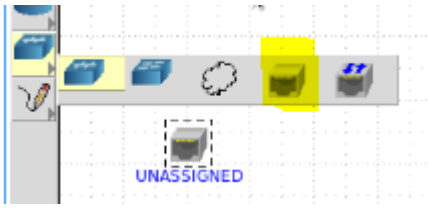


Fig. 2 The CORE RJ-45 Component

### 2.3.3 XenServer and CORE Integration

A VM running on XenServer can have up to 7 virtual Ethernet interfaces. The XenCenter software can be used to configure these interfaces using the networking tab (shown in Fig. 3). These interfaces are used to communicate among virtual machines in the XenServer software.

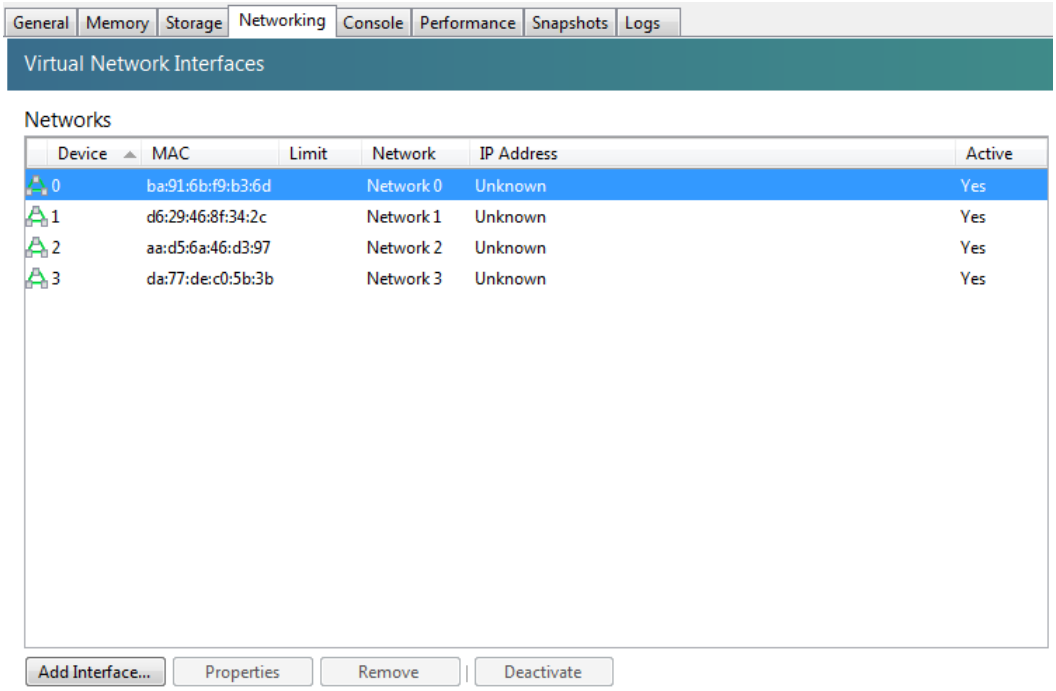


Fig. 3 Network devices on XenServer virtual machine

XenServer allows traffic isolation by binding virtual interfaces to specific networks (see Fig. 4). These networks are tied to the physical NIC on the host machine (in our case, the Dell PowerEdge M820 consisted of 4 physical NICs). This feature enables traffic isolation; that is, several instances of the same network infrastructure can be executed simultaneously without any overlap.

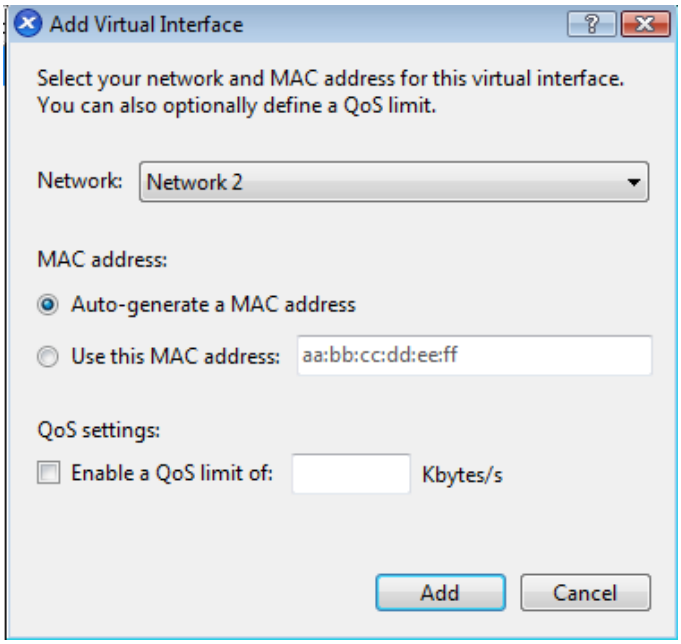


Fig. 4 Adding a network device to a virtual machine in XenServer

When a virtual machine is started, XenCenter also has a built-in virtual desktop system under the console tab. The physical or the Media Access Control (MAC) addresses associated with the virtual network interfaces can be viewed from the networking tab of XenCenter or within the running VM. This MAC address can be used to configure specific Dynamic Host Configuration Protocol (DHCP) servers to respond only to specific machines.

**Configuring DHCP**

DHCP servers can be run from within routers in CORE scenarios. These DHCP servers are configured to provide an Internet Protocol (IP) address to machines with specific MAC addresses. This facilitates subnet association of the virtual machines in CORE scenarios. In CORE, we executed the following command to install the dhcp server:

```
apt-get install isc-dhcp-server
```

In the CORE GUI, right-clicking on a router displays the options shown in Fig. 5. Choosing the *Configure* menu item shows the window in Fig. 6.

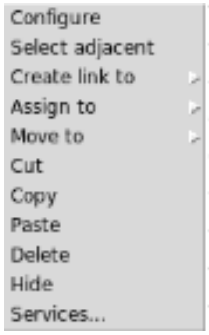


Fig. 5 Router node options selection menu

Each DHCP server is responsible for a subnetwork in the CORE scenario. The router must have an IP address within the subnetwork that will be served. To enable the DHCP on startup, the *Services* button on the *router configuration* window was selected.

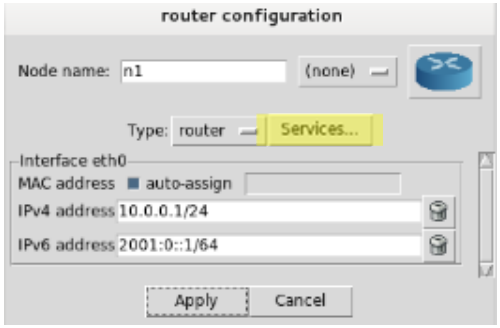


Fig. 6 Router node-interface configuration screen

Next, the DHCP button is pressed on the router *Node services* configuration screen window to enable the service. The tool icon next to the service is selected to configure the DHCP server (Fig. 7).

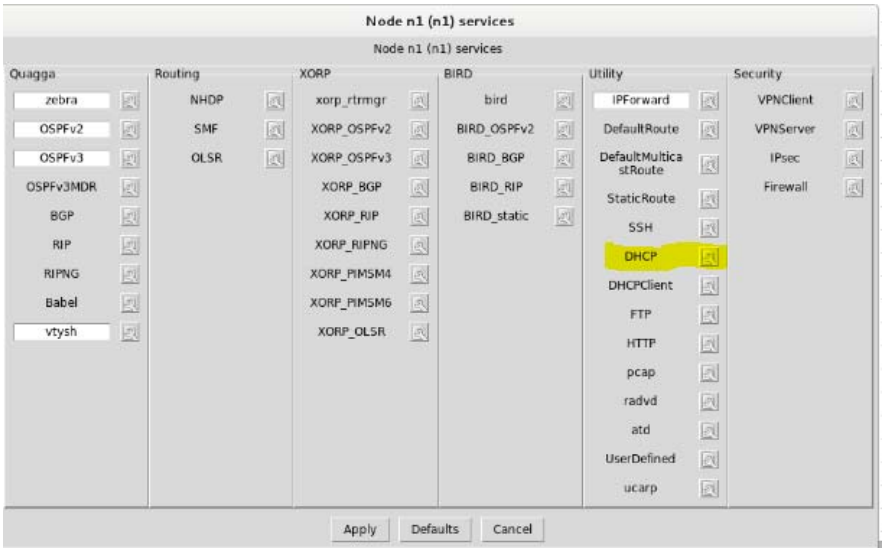


Fig. 7 Router node-services configuration screen

Figure 8 shows an example configuration that assigns the IP address 192.168.1.83 to the VM with the MAC address AE:ED:D4:20:39. There are many options to configure. (Use the DHCP hyperlink[30] to view a more complete description of the dhcpd.conf file.)

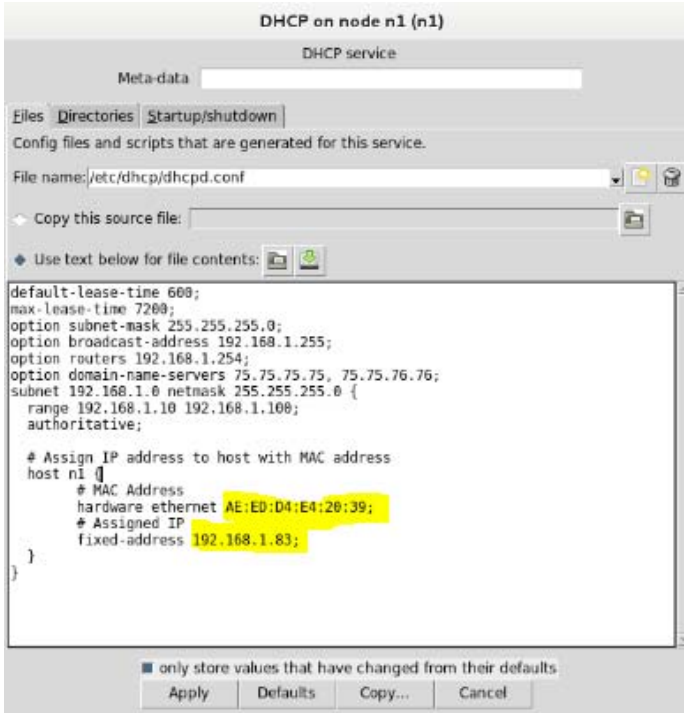


Fig. 8 CORE router-node DHCP configuration

### 2.3.4 Virtual Ethernet Node Setup

Right-clicking on the icon shows a listing of menu items; clicking on the *Configuration* menu item allows the user to setup the RJ-45 component to communicate to external entities (see Fig. 9).

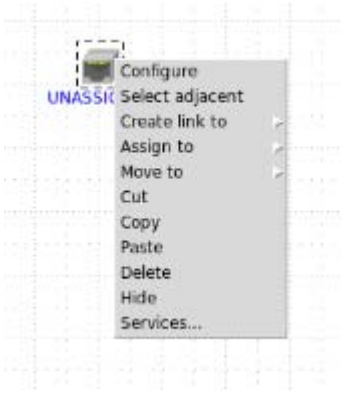


Fig. 9 CORE RJ-45 component options menu

The RJ-45 component needs to be associated with a network interface card through which it will send and receive network traffic. Figure 10 shows the interfaces that may be assigned in the case of our particular CORE VM running in XenServer.

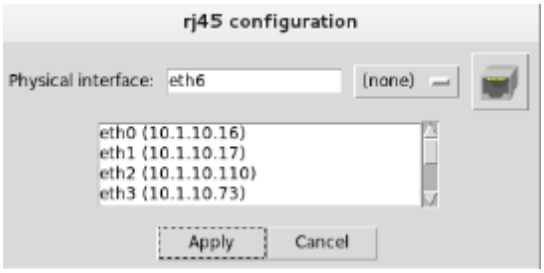


Fig. 10 RJ-45 component interface-selection menu

In Kali Linux, the default network manager constantly attempts to pull DHCP data from a server on the network. This can interfere with CORE. To prevent this, the *Disconnect* menu item should be pressed for each network interface used with CORE from Kali's network-manager interface list (see Fig. 11).

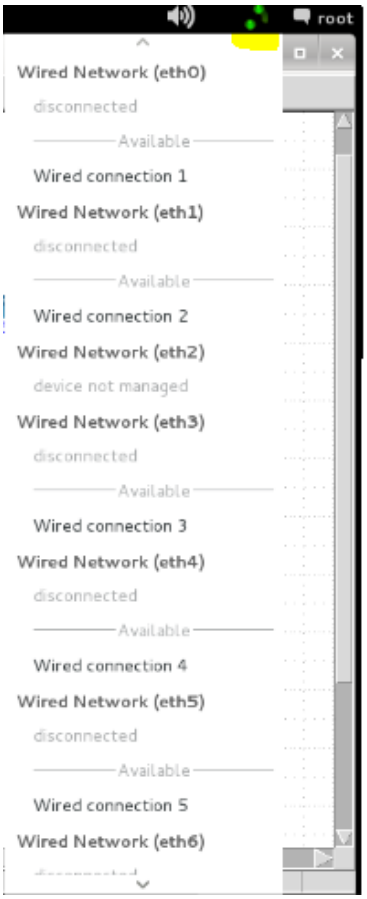


Fig. 11 Kali Linux network-manager interface list

### 2.3.5 XenServer Startup and Shutdown from within CORE

The XenServer VMs are automatically reverted to snapshots and started by scripts using the Python back end: XenAPI. This allows the user to click the start/stop button to start not only CORE but all XenServer virtual machines associated with the scenario.

First, the hooks item from the session menu in CORE is selected (see Fig. 12).

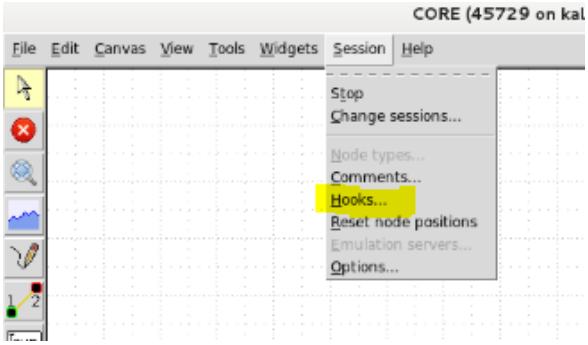


Fig. 12 CORE-session menu

A new hook is added (Fig. 13).

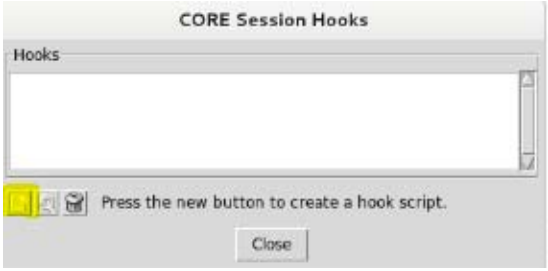


Fig. 13 CORE-session hooks menu

Both a startup script (by selecting configuration in the drop-down window) and a script that executes when the scenario is terminated (by selecting *Shutdown* in the drop-down window) are created. See Figs. 14 and 15.

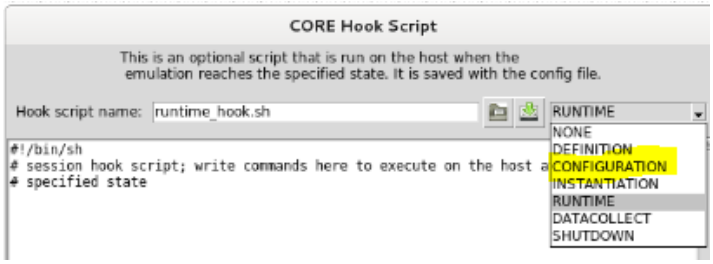


Fig. 14 CORE hook-script configuration script

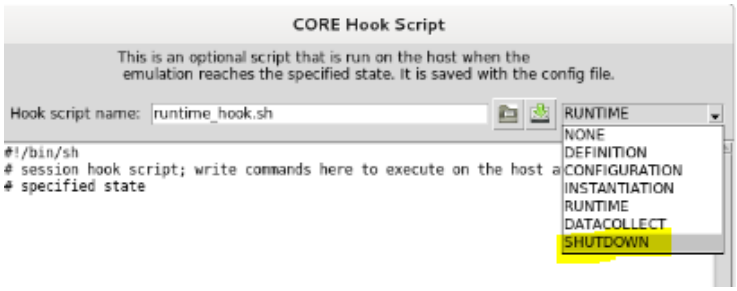


Fig. 15 CORE hook-script shutdown script

The *Configuration* CORE hook script calls startup.py, with the following syntax:

`startup.py <snapshot_name> <vm_name>`

In our case, we created a virtual machine named sn1 and captured a snapshot named sn1hw3 (see Fig. 16). The entire contents of shutdown.py can be seen in the Appendix of this report.

The startup script reverts the VM named <vm_name> to a snapshot named <snapshot_name> and then starts the VM from where the snapshot was taken. A snapshot taken of the disk and

memory will start without needing to go through the boot process. The script requires the VM and snapshot names to be unique.

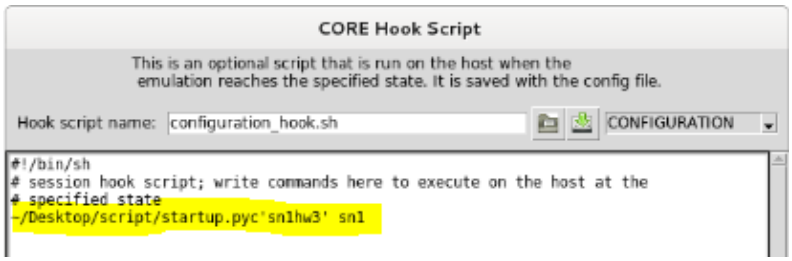


Fig. 16 CORE-configuration script content

The *Shutdown* CORE hook script calls shutdown.py, with the following syntax:

`shutdown.py <vm_name>` (see Fig. 17)

The entire contents of shutdown.py can be seen in the Appendix. This script terminates all VMs with the given name.

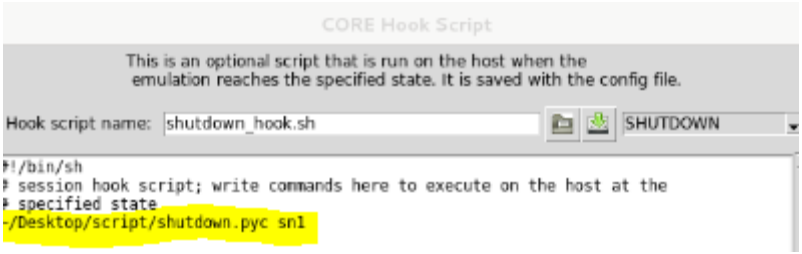


Fig. 17 CORE-shutdown script content

#### 2.3.6.1 XenServer Login Note

Both scripts login to the management interface of XenServer. In production use, the credentials should not be hard-coded into the source files as they can be easily read by a third party.

#### 2.3.6.2 XenAPI.py

The XenAPI.py file is required by both the startup.py and shutdown.py scripts and should be located in the same directory as the scripts. Alternatively, the XenAPI.py can be placed in the Python path.

---

## 3. Case Study—Building an Exploitable Scenario

---

We built a small capture-the-flag scenario to demonstrate the testbed's functionality. First, we created the network topology using CORE. Next, we installed several vulnerable Windows

virtual machines in XenServer. Lastly, we configured CORE to communicate with XenServer to automatically start and stop virtual machines.

### 3.1 The CORE scenario

The CORE scenario consists of 3 subnetworks: 10.0.0.0/24; 10.0.1.0/24; and 10.0.2.0/24. The nodes on the network are as follows:

- Four generic Linux nodes: One node is a simple workstation (n1, no services running); 2 are routers (n2/n3, using the “open shortest path first” routing protocol and running DHCP servers); and one is a server (n8, running an SSH server).
- Two RJ-45 hardware-in-the-loop interfaces: One is used for a XenServer VM running Windows (eth1, running an ftp server), and the other is an entry point for a penetration tester (eth0, running the Kali-Linux operating system on a laptop).

Figure 18 shows our network topology.

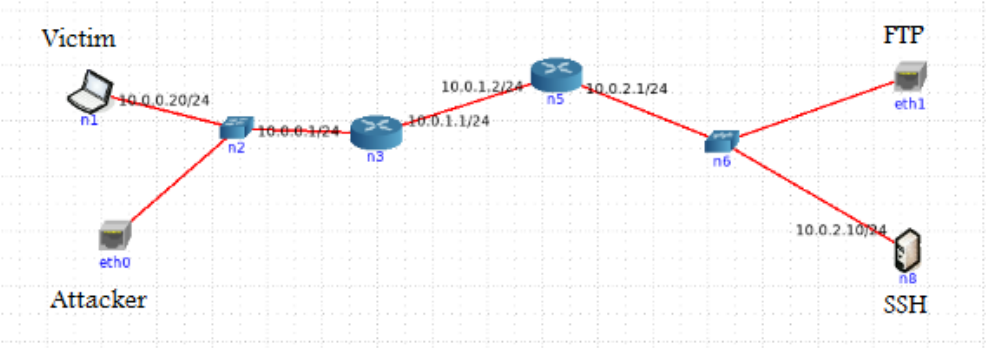


Fig. 18 CORE-network scenario

In this scenario an attacker steals a victim’s file transfer protocol (FTP) login credentials. A script simulating a user at the victim machine will log into the FTP server and reconnect when disconnected.

The attacker is expected to map the network and discover open ports. The attacker should then identify—using a man-in-the-middle attack—that the client has an active FTP connection with a server on a remote subnetwork. Interrupting the FTP connection will force the victim to reconnect; monitoring this traffic will reveal the login credentials in plaintext. These credentials can then be used to impersonate the client and connect to the FTP server leading to full account disclosure. This scenario demonstrates the risks of sending passwords in the clear over an insecure network.

### 3.2 Setting up FTP on a XenServer VM

The EasyFTP server was installed on a Windows 7 virtual machine. Afterward, 2 users were configured and the FTP daemon was started (Figs. 19 and 20).

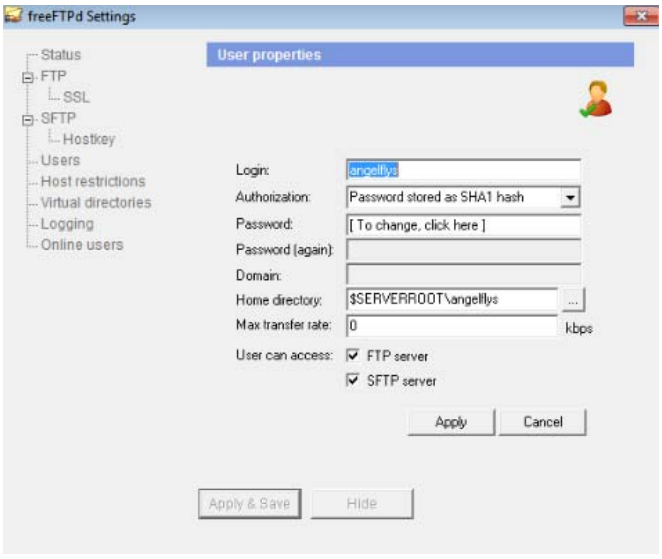


Fig. 19 Creating 2 accounts in the EasyFTP server GUI

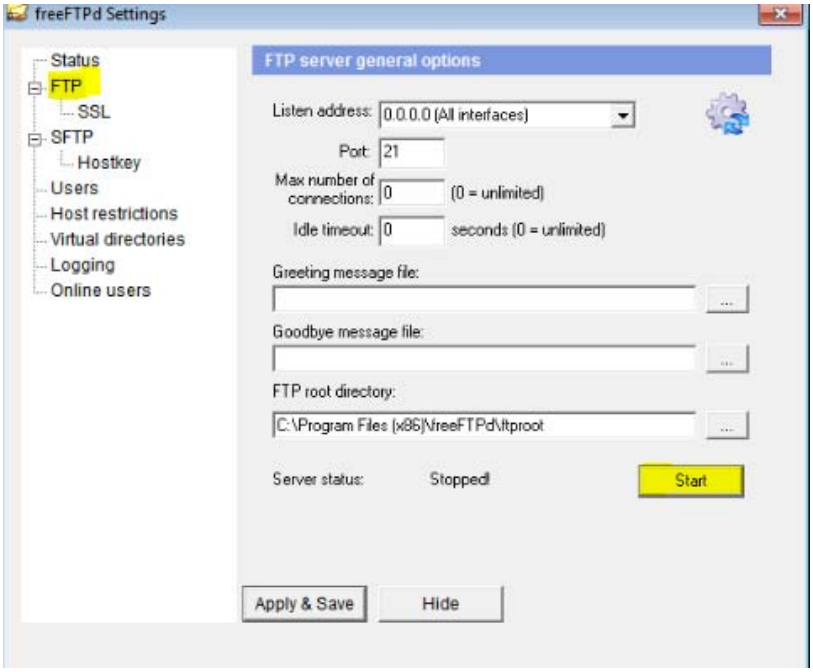


Fig. 20 Starting the Easy FTP daemon

To start the FTP server on system boot, we placed a shortcut to the EasyFTP executable in the startup folder of the Windows 7 VM (Fig. 21).

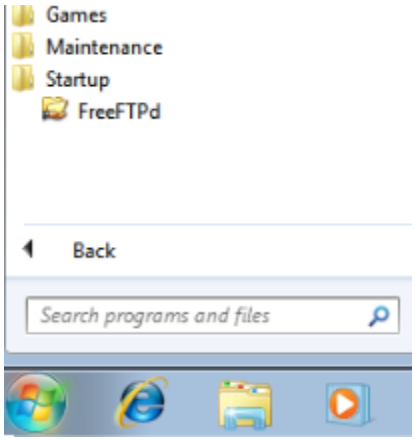


Fig. 21 Startup shortcut to FreeFTPd

After the FTP server was configured, we captured a snapshot and named it sn1hw3 (see Fig. 22).

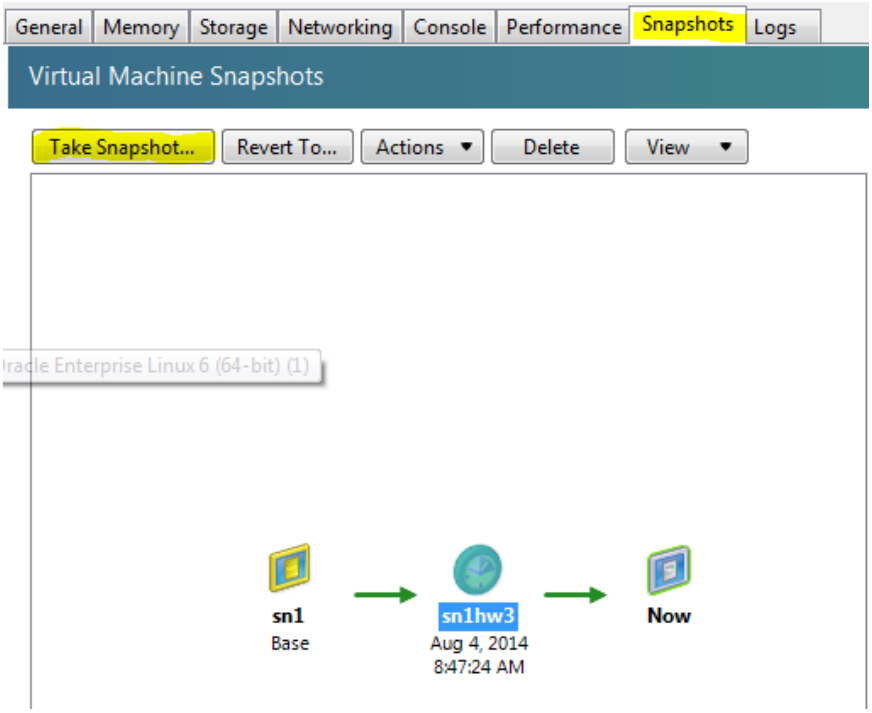


Fig. 22 Runtime snapshot for a Windows FreeFTPd server in XenServer

### 3.3 Assigning the FTP Server to a Subnetwork in CORE using DHCP

In this CORE scenario, there are 2 routers running the dhcp service. This could result in a conflict when XenServer virtual machines attempt to pull DHCP information with default settings, as either of the DHCP server may respond (causing a race condition). For this reason, we configured the router DHCP services to respond only to machines with specific MAC addresses. The router at 10.0.2.1/24 only supplies an IP address to the FTP machine (in this case, with MAC address FA:8F:7E:1D:11:22). The resulting dhcpd.conf file on the 10.0.2.1/24 router

can be seen in Fig. 23. All other machines on the network must statically assign an IP address. The attacker must either supply his/her MAC address to the 10.0.0.1/24 router in CORE or statically assign an IP address in the 10.0.0.x subnetwork.

```
# auto-generated by DHCP service (utility.py)
# NOTE: move these option lines into the desired pool { } block(s) below
#option domain-name "test.com";
#option domain-name-servers 10.0.0.1;
#option routers 10.0.0.1;

log-facility local6;

default-lease-time 600;
max-lease-time 7200;

ddns-update-style none;

subnet 10.0.2.0 netmask 255.255.255.0 {
  host sn1 {
    hardware ethernet FA:8F:7E:1D:11:22;
    fixed-address 10.0.2.20;
  }
}
```

Fig. 23 DHCP configuration file on a CORE router node

### 3.4 Executing an FTP Client

We hosted an ftp client on node n1 in CORE by assigning the script, conn.py, to run on boot. The entire contents of this script are included in the Appendix.

### 3.5 Penetration-Test Walkthrough

The following is one possible sequence of commands that could be executed by a penetration tester to steal the credentials from the FTP traffic:

1. The attacker starts a network sniffer to view all traffic that is broadcast/multicast on the local subnetwork.

   Command:
   ```
   wireshark &
   ```

   Result: Only address resolution protocol (ARP) traffic is seen because the attacker is on a switched network.

2. The attacker maps the network and looks for any open ports that would indicate potentially vulnerable services.

   Command:
   ```
   nmap -n 10.0.0.0/24
   ```

   Result: only one other host in the attacker's local subnetwork (victim). This machine is not running any services.

3. The attacker executes a man-in-the-middle attack. The attacker advertises their MAC address as the default gateway. In this case, the gateway is using the first available IP address in the subnetwork (this is very common). In order to view bidirectional traffic, the attacker must

also advertise themselves as the victim machine. In addition, the attacker must also set a kernel option to enable traffic forwarding; otherwise, this will cause a denial of service between the victim and gateway.

Commands:

```
sysctl -w net.ipv4.all.conf.forwarding=1
arpspoof -t 10.0.0.20 10.0.0.1
arpspoof -t 10.0.0.1 10.0.0.20
```

Result: The attacker can now observe all traffic between the victim machine and the router.

4. At this point the attacker is able to see all traffic; however, the credentials are not in the capture because the connection is currently ongoing. The attacker must kill the connection and force the victim to re-authenticate to the FTP server.

   Command:

   ```
   tcpkill -i eth0 port 21
   ```

   Result: After killing all tcp connections on Port 21, the victim loses connectivity to the FTP server. The user logs back in (this is programmed into the conn.sh script—see the Appendix for the contents of this script). Because all traffic now passes through the attacker's machine, the login username and password are now made available to the attacker.

---

# 4. Lessons Learned

---

## 4.1 Adding Interfaces for Use with CORE RJ-45 Components

We needed a scalable way to add external hardware (switches, routers, laptops, etc.) and external software (other XenServer virtual machines, software routers, etc.) to our CORE scenarios. The RJ-45 components are used for this purpose; however, once bound to the RJ-45 component, the physical interface can no longer be used elsewhere. In addition, if traffic is meant to be isolated from other RJ-45 components, a unique physical interface must be assigned to the different RJ-45 components. For example, in the scenario shown in Fig. 24 there are 2 subnets connected to two RJ-45 interfaces (both are bound to the same physical interface: eth0). In this case, the traffic will not be isolated; nodes in the 10.0.0.1/24 network can see the traffic in the 10.0.1.1/24 network and vice versa.

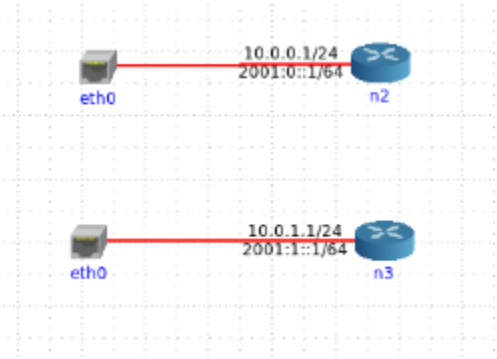


Fig. 24 Two CORE RJ-45 components bound to the same physical interface

To overcome this limitation, the 2 RJ-45 components must be assigned to different physical interfaces. Linux offers several options for creating virtual interfaces; however, none of these methods worked effectively (and will be described next).

### 4.1.1 Failed Attempts at Adding Multiple RJ-45 Components Bound in CORE

We tried several mechanisms to add virtual interfaces from within a Linux environment. All of these methods relied on using an existing physical network interface and creating virtual interfaces at different layers of the network stack. We used the scenario shown in Fig. 25.

We use *ip link show* to view the interfaces created when the CORE software is started. Each virtual interface will have a new interface created for the node.

```
5: eth3: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc pfifo_fast state UP mo
de DEFAULT qlen 1000
    link/ether 0e:bd:91:5d:9f:c3 brd ff:ff:ff:ff:ff:ff
6: eth4: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc pfifo_fast state UP mo
de DEFAULT qlen 1000
    link/ether 96:30:14:cd:43:8f brd ff:ff:ff:ff:ff:ff
7: eth5: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc pfifo_fast state UP mo
de DEFAULT qlen 1000
    link/ether ba:d2:62:36:11:3e brd ff:ff:ff:ff:ff:ff
8: eth6: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc pfifo_fast state UP mo
de DEFAULT qlen 1000
    link/ether ce:2d:57:3e:71:ec brd ff:ff:ff:ff:ff:ff
120: b.41580.46892: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc noqueue sta
te UP mode DEFAULT
    link/ether 22:a7:8e:82:66:89 brd ff:ff:ff:ff:ff:ff
122: n2.eth0.155: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc pfifo_fast ma
ster b.41580.46892 state UP mode DEFAULT qlen 1000
    link/ether de:fe:9e:64:08:df brd ff:ff:ff:ff:ff:ff
123: b.63592.46892: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc noqueue sta
te UP mode DEFAULT
    link/ether a6:0f:8b:e9:49:6a brd ff:ff:ff:ff:ff:ff
125: n3.eth0.155: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc pfifo_fast ma
ster b.63592.46892 state UP mode DEFAULT qlen 1000
    link/ether a6:0f:8b:e9:49:6a brd ff:ff:ff:ff:ff:ff
```

Fig. 25 A screenshot of some of the interfaces created by CORE

In this case, the interface named n3.eth0.155 is the eth0 interface for the n3 node and n2.eth0.155 is the eth0 interface for the n2 node in CORE (see Fig. 26).

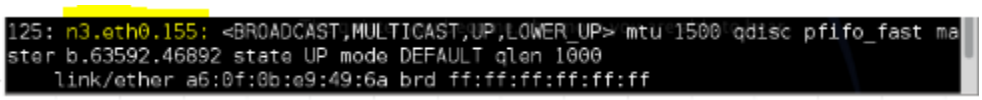


Fig. 26 A screenshot of the interface created for node n3 in CORE

We use *brctl show* to view the bridges created during CORE startup. In addition to the virtual ethernet interfaces, a bridge is also created for each interface: b.41580.46892 and b.63592.46892 (see Fig. 27). Then, the interface is added to the bridge. Unfortunately, the physical interface eth0 can be assigned to only one bridge.


Fig. 27 The bridges created by CORE

Several attempts were made to get around this limitation. One way was to move both interfaces to the same bridge. The bridge without eth0 then becomes unused.

```
brctl delif b.63592.46892 n3.eth0.155
brctl addif b.41580.46892 n3.eth0.155
```

While both routers are able to send data through eth0, the traffic is also observable from both routers in CORE.

Our team used the IP command to create several different link types such as vlan, veth, dummy, macvlan, bridge, etc. (e.g., *ip link add link eth0 eth0.1 type macvlan*). We attempted to use all of these links to connect Ethernet nodes in CORE to eth0. However, each had its own set of problems and did not work. For example, when using a macvlan the traffic will flow out but not back. ARP requests pass through eth0 and the destination responded but never arrived on the macvlan link. We suspect this is because the MAC address of eth0 and the node inside CORE are different. However, even by making the MAC address the same in CORE as eth0 failed in the same manner. While we are confident there are other solutions to this issue, the most elegant and efficient solution we encountered was to create a CORE VM in XenServer and add several vifs. These were then used inside the CORE VM as if they were truly physically present network interfaces.

### 4.1.2 Implemented Solution

Ultimately, in order to add multiple external RJ-45 interfaces with isolated traffic, we created a CORE virtual machine in XenServer. We then added several virtual interfaces to the CORE VM with XenServer. At first, this solution was not immediately obvious as the XenCenter graphical interfaces limit the number of interfaces to 7. Our team overcame this limitation by installing XenTools on the CORE VM and using the vif-create command in the XenServer command-line interface.

### 4.2 Broadcast Packets Are Not Isolated

A limitation of using CORE using XenServer virtual interfaces as entry points for virtual machines exists when broadcast packets (e.g., ARP requests) are transmitted by entities in the CORE scenario. In the scenario shown in Fig. 18, an attacker will position themselves in the network using the RJ-45 eth0 interface. The FTP server machine is on a separate subnetwork, with its entry point using the RJ-45 eth1 interface. In a real network scenario, no messages could traverse these 2 subnetworks without passing through the appropriate routers. However, the way our testbed is implemented the attacker will be able to see ARP requests made by any machine on the second subnet. This is because there is nothing in software or hardware that is isolating layer-2 traffic.

The physical equivalent to this scenario is having a machine with multiple physical interfaces, but all of the interfaces are using the same switch. While the traffic is isolated based on IP address, any packets that are broadcast will reach all nodes that are connected to the switch.

#### 4.2.1 Implemented Solution

Our workaround for this issue is to create a XenServer internal network that will be used by all XenServer VMs. The CORE virtual machine will be connected to the internal network as well as the physical external network which is then connected to a switch. The penetration tester can then connect to the switch and have an isolated view of the subnetwork.

## 5. Conclusions

We have developed a penetration-testing environment that can be used to facilitate creation of capture-the-flag scenarios. What makes this ARL testbed novel, however, is that it allows the creation of complex (consisting of several platforms, networking components, etc.) pentest-training scenarios that can be saved as CORE .imn or .xml files and easily retrieved at a later time. Within this environment, penetration-testing organizations can create custom scenarios that mimic fielded scenarios and use them as many times as needed to train, improve, and analyze group skill sets.

Moreover, the work described in this technical report is part of a larger, longer-term effort whose goal is the creation of a decision-guidance system for penetration testers. Many times the quality of a penetration test depends on the experience levels of the personnel involved. With a decision-guidance system, testers will be given advice depending on the current network's state that will portray actions of previous testers in similar circumstances. Our team took the first step toward this goal by creating this testbed. The next step is to create tools for capturing testers' actions depending on the state of the network. Afterward, we intend to develop metrics to determine best

courses of action (realizing that a tester may perform better due to the use of a particular tool). Lastly, we plan to derive rule sets based on these metrics to guide testers in future assessments.

## 6. References


1. Directive 8570.01. Information assurance training, certification, and workforce management. Department of Defense (US); 2004 Aug 15.
2. CISSP. International Information Systems Security Certification Consortium; 2015 [accessed 2015 Jan 5]. https://www.isc2.org/CISSP/Default.aspx
3. CISA. ISACA; 2015 [accessed 2015 Jan 5]. http://www.isaca.org/Certification/CISA-Certified-Information-Systems-Auditor/Pages/default.aspx
4. CEH. International Council of Electronic Commerce Consultants; 2015 [accessed 2015 Jan 5]. http://www.eccouncil.org/Certification/certified-ethical-hacker
5. NVD. National Institute of Standards and Technology; 2015 [accessed 2015 Jan 5]. http://nvd.nist.gov
6. MITRE Common Vulnerabilities and Exposures. MITRE; 2015 [accessed 2015 Jan 5]. http://www.rapid7.com/db/
7. VirtualBox user manual. Oracle; 2015 [accessed 2015 Jan 5] https://www.virtualbox.org/manual/UserManual.html
8. Rosenblum M. VMware's virtual platform: A virtual machine monitor for commodity PCs. Proceedings of the Hot Chips 11, Symposium on High Performance Chips; 1999 Aug. p. 185–196.
9. Williams D, Garcia J. Virtualization with Xen: including Xenenterprise, Xenserver, and Xenexpress. Syngress Publishing, Inc. 2007.
10. Spitzner L. The honeynet project: Trapping the hackers. IEEE Sec Priv. 2003;1(2):15–23.
11. The exploit database; 2015 [accessed 2015 Jan 5]. http://www.exploit-db.com/
12. Vulnerability and exploit database. Rapid 7; 2015 [accessed 2015 Jan 5]. http://www.rapid7.com/db/
13. Maynor D. Metasploit toolkit for penetration testing, exploit development, and vulnerability research. Elsevier Inc. 2011.
14. Weidman G. Penetration testing: A hands-on introduction to hacking. No Starch Press. 2014.

15. Tools: Unicorn.py – Using Metasploit to powershell. Offensive Security Blog V2.0; 2015 [accessed 2015 Jan 5]. http://www.r00tsec.com/2014/08/tools-unicornpy-using-metasploit-to.html

16. Oberhumer MFXJ, Molnár L, Reiser J. UPX: Ultimate packer for eXecutables; 2014 [accessed 2015 Jan 5]. http://upx.sourceforge.net

17. Infosec Institute; 2015 [accessed 2015 Jan 5]. http://www.infosecinstitute.com/

18. Defcon capture the flag. DEF CON Hacking Conference 2015; [accessed 2015 Jan 5]. https://www.defcon.org/html/links/dc-ctf.html

19. Capture the flag challenge. University of Connecticut Comcast Center of Excellence for Security Innovation; 2014 [accessed 2015 Jan 5]. http://www.csi.uconn.edu/network-challenge

20. International capture the flag. University of California at Berkley; 2015 [accessed 2015 Jan 5]. http://ictf.cs.ucsb.edu/#/

21. Kopan T. Hacking contests ID cyber talent for government, industry. Politico; 2015 [accessed 2015 Jan 5]. http://www.politico.com/story/2014/07/hacking-contests-id-cyber-talent-for-government-industry-109494.html

22. Freckleton R. Annual capture the flag cyber challenge. MITRE; 2014 [accessed 2015 Jan 5]. http://www.mitre.org/capabilities/cybersecurity/overview/cybersecurity-blog/annual-capture-the-flag-cyber-challenge

23. Mirkovic J, Terry B. Teaching cybersecurity with DeterLab. IEEE Sec Priv 2012; 10(1): 73–76.

24. CTF365–Information security through gamification. HackaServer; 2014 [accessed 2014 October 5]. https://hackaserver.com/

25. Ahrenholz J, Danilov C, Henderson TR, Kim JH. CORE: A real-time network emulator. Proceedings of the Military Communications Conference; 2008 Nov. p. 1–7.

26. Ahrenholz J, Goff T, Adamson B. Integration of the CORE and EMANE network emulators. Proceedings of the Military Communications Conference; 2011 Nov. p. 1870–1875.

27. Open source virtualization. XenServer; 2015 [accessed 2015 Jan 5]. http://XenServer.org/

28. Official Kali Linux downloads. Kali Linux; 2015 [accessed 2015 Jan 5]. http://www.kali.org/downloads/

29. Common open research emulator. Network and Communication Systems Branch of the Naval Research Laboratory; 2015 [accessed 2015 Jan 5]. http://www.nrl.navy.mil/itd/ncs/products/core

30. Linux man page for dhcpd.conf. die.net; 2015 [accessed 2015 Jan 5]. http://Linux.die.net/man/5/dhcpd.conf

INTENTIONALLY LEFT BLANK.

## Appendix: Contents of Startup, Shutdown, and FTP-Client Scripts

This appendix appears in its original form, without editorial change.

The following are the contents of the startup.py script.

**Startup Script**

```
import sys
import XenAPI

url = 'https://12.0.0.201'    # URL of Xen Server
uname = 'root'    # XenServer login Username
pwd = '##########  ' ' pwd

# Revert <vm_name> VM to <snapshot_name> snapshot and start VM
# Assumes vm_name and snapshot_name are unique.
def start_server(session, snapshot_name, vm_name):
      # Revert to snapshot
      vm = session.xenapi.VM.get_by_name_label(snapshot_name)
      print 'Reverting ...',
      sys.stdout.flush()
      session.xenapi.VM.revert(vm[0])
      print 'Done'

      # Resume suspended VM after reverting to snapshot
      vm = session.xenapi.VM.get_by_name_label(vm_name)
      print 'Resuming ...',
      sys.stdout.flush()
      session.xenapi.VM.resume(vm[0], False, True)
      print 'Done'


# parse cmd args, password exposed
def parm_py():
      global url, uname, pwd, snapshot_name, vm_name
      # Invalid parameters, display useage
      if len(sys.argv) < 4:
            print "Usage:"
            print sys.argv[0], "<url> <username> <password>
[snapshot_name][<vm_name>]"
            sys.exit(1)
      url = sys.argv[1]
      uname = sys.argv[2]
      pwd = sys.argv[3]
      snapshot_name = sys.argv[4]
      vm_name = sys.argv[5]

# Parse cmd args, pwd hidden in .py
def parm_pyc():
      global snapshot_name, vm_name
      if len(sys.argv) < 1:
            print "Usage:"
            print sys.argv[0], "<snapshot_name> <vm_name>"
            sys.exit(1)
      snapshot_name = sys.argv[1]
      vm_name = sys.argv[2]

# To avoid exposing login credentials to XenServer the script
# can be compiled with
#    python -m py_compile <script_name>
```

```
# creating a .pyc file.
parm_pyc()  # Use parm_pyc() with .pyc, or parm_py() for .py
print "Connecting... ",
sys.stdout.flush()
session = XenAPI.Session(url)
session.xenapi.login_with_password(uname, pwd)
print 'Done'
try:
      start_server(session, snapshot_name, vm_name)
finally:
      session.xenapi.session.logout()
```

The following are the contents of the shutdown.py script.

**Shutdown Script**

```
import sys
import XenAPI

url = 'https://12.0.0.201'    # URL of Xen Server
uname = 'root'    # XenServer login Username
pwd = '$umm3r2014'      #  ' ' pwd
vm_name = ''

# Force <vm_name> to shutdown
def stop_server(session, vm_name):
      # Shutdown each VM named vm_name
      vms = session.xenapi.VM.get_by_name_label(vm_name)
      for vm in vms:
            # Shutdown VM if running
            record = session.xenapi.VM.get_record(vm)
            upstate = record["power_state"]
            print "VM %s is %s: " % (vm_name, upstate),
            sys.stdout.flush()
                if upstate == "Running":
                  print "Stopping ...",
                  sys.stdout.flush()
                        session.xenapi.VM.hard_shutdown(vm)
                  print "Done"


# Parse cmd args, password exposed
def parm_py():
      global url, uname, pwd, vm_name
      if len(sys.argv) < 4:
            print "Usage:"
            print 'Shutdown a scenario'
            print sys.argv[0], "<url> <username> <password> [<vm_name>]"
            sys.exit(1)
      url = sys.argv[1]
      uname = sys.argv[2]
      pwd = sys.argv[3]
      vm_name = sys.argv[4]

# Parse cmd args when pre-compiled to hide password
```

```
def parm_pyc():
      global vm_name
      if len(sys.argv) < 1:
              print "Usage:"
              print 'Shutdown a scenario'
              print sys.argv[0], "<vm_name>"
              sys.exit(1)
      vm_name = sys.argv[1]

# To avoid exposing login credentials to XenServer the script
# can be compiled with
#    python -m py_compile <script_name>
# creating a .pyc file.
parm_pyc()  # Use parm_pyc() with .pyc, or parm_py() for .py
print "Connecting ...",
sys.stdout.flush()

# Login to XenServer
session = XenAPI.Session(url)
session.xenapi.login_with_password(uname, pwd)
print 'Done'
try:
      stop_server(session, vm_name)
finally:
      session.xenapi.session.logout()
```

The following are the contents of the conn.py script.

**FTP client script: conn.sh**

```
from ftplib import FTP
import time, socket, sys
def ftpConnect():
      try:
            ftp = FTP('10.0.2.11',####,####) //ip address, username, password
            while True:
                  ftp.cwd('')
                  time.sleep(5)
            except socket.error:
                  print "caught socket.error"

while True:
      ftpConnect()
      print "sleeping for 5 seconds"
      time.sleep(5)
```

## List of Symbols, Abbreviations, and Acronyms

| | |
|---|---|
| API | application programming interface |
| ARL | Army Research Laboratory |
| ARP | address resolution protocol |
| CEH | Certified Ethical Hacker |
| CISA | Certified Information Systems Auditor |
| CISSP | Certified Information Systems Security Professional |
| CORE | common open research emulator |
| DHCP | Dynamic Host Configuration Protocol |
| DOD | Department of Defense |
| EMANE | Extendable Mobile Ad Hoc Network Emulator |
| FTP | File Transfer Protocol |
| GUI | graphical user interface |
| IP | Internet Protocol |
| ISACA | Information Systems Audit and Control Association |
| MAC | Media Access Control |
| NIC | Network Interface Card |
| NVD | National Vulnerability Database |
| SSH | Secure Shell |
| VMs | Virtual machines |

1 (PDF) DEFENSE TECHNICAL INFORMATION CTR DTIC OCA

2 (PDF) DIRECTOR US ARMY RESEARCH LAB RDRL CIO LL IMAL HRA MAIL & RECORDS MGMT

1 (PDF) GOVT PRINTG OFC A MALHOTRA

CHIEF
US ARMY RESEARCH LAB
RDRL SLE
RICHARD FLORES

CHIEF
US ARMY RESEARCH LAB
RDRL SLE I
DANIEL W LANDIN

CHIEF
US ARMY RESEARCH LAB
RDRL SLE A
EDWARD L ZARRET